THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 61, No. 10, December 1982
Printed in U.S.A.

Design of Recovery Strategies for a Fault-Tolerant No. 4 Electronic Switching System

By R. J. WILLETT

(Manuscript received April 13, 1982)

This paper focuses on the design of recovery strategies that have been developed to give the No. 4 Electronic Switching System (ESS) the necessary high level of fault tolerance. Reliability and availability are the key watchwords to operational integrity in the No. 4 ESS. It is the world's largest toll and tandem switching system, capable of handling 616,000 call attempts per hour, and serving up to 100,000 active terminations. With a downtime objective of less than two hours in 40 years, the No. 4 ESS must be designed to be very fault tolerant viewed from the caller's perspective.

I. INTRODUCTION

The No. 4 Electronic Switching System (ESS) is a high-capacity toll and tandem digital and switching system that is capable of handling 616,000 call attempts per hour and serving up to 100,000 active terminations.

The maintenance system in the No. 4 ESS has the responsibility of responding to error conditions reported by hardware-fault-detection circuits, by memory mutilation detectors, or by some other system-integrity monitor. The strategies invoked for rapid isolation of faulty configurable entities or for correcting memory errors are based on the type of error condition, the state of the system at the time of the failure, and the history of previous recovery actions that may have been attempted.

The recovery strategies employed are fundamentally two-dimensional. One dimension is concerned with the present and the other with the past. Probably the most powerful and unique property of the No. 4 ESS fault recovery system is the latter dimension, that of

considering the past, in a process called error analysis. Much more will be said of this process later in this paper.

Although the basic mission of the maintenance recovery system in the No. 4 ESS has remained the same throughout its evolution, the maintenance recovery strategies developed during that evolution have been influenced by many external factors. The availability of new hardware technologies and the development of redesigned and cost-reduced hardware have created the need for making changes in existing strategies. Revisions dictated by field experience and by the introduction of new features have proved emphatically the necessity of being adaptable to change.

This paper discusses the development of fault-tolerant strategies in general, and the state of the art as applied to the No. 4 ESS.

II. FAULT TOLERANCE OBJECTIVE

As we stated earlier, the No. 4 ESS is a large toll and tandem switch capable of handling 616,000 call attempts per hour and serving up to 100,000 active terminations. With a downtime criterion of less than two hours in 40 years, the No. 4 ESS must be, of necessity, a highly reliable system. Recovery actions must be carried out as quickly as possible, and with minimum disturbance to calls active in the system. To put it succinctly, the No. 4 ESS system should be available for processing calls at all times. Any hardware failure or memory error should be detected immediately, before it has a chance to cause call mishandling. From a practical standpoint, this objective cannot be realized under all circumstances, but nonetheless, it is the guiding principle in the design of the No. 4 ESS fault-tolerant structure. Minimizing recovery time and service impact are paramount. Recovery time following the detection of a hardware fault should be measured in terms of 100 milliseconds or less, including the process of detecting, isolating, and restarting the interrupted program. Most faults should be resolved with only one interrupt stimulus. Transient faults (with various manifestations such as transient, intermittent, marginal) may require several interrupts to resolve. Also, multiple fault situations can be experienced, and may take several interrupts for resolution. When such multiple faults can cause limited service impairment, the recovery system must be capable of degrading that service gracefully, i.e., confining the effects to the functional area directly affected.

Memory integrity failures should be corrected by reconstructing valid data from associated information, or if that is not possible, then by reinitializing the memory structure. The former causes less system perturbation, but takes longer; the latter is faster, but can cause more service impact. The choice is usually dictated by the type of structure and by real-time considerations.

III. SYSTEM ARCHITECTURE

Fault tolerance implies system survival under fault conditions. Survival requires alternatives. The architecture employed in the No. 4 ESS is so varied, and the interconnection of components and subsystems is so flexible that several strategies had to be used to achieve the necessary fault tolerance. The design of those recovery strategies takes into consideration system objectives, the hardware technology, system environment, failure rates and failure modes, and the software structure of both the operational and maintenance systems.

3.1 Hardware structure

The basic communications format in the processor and in most of the periphery uses all seems well (ASW) and/or all seems well failure (ASWF) conventions, and single and multibit parity protection over instruction and data transmission. Generally, unique error indicators, such as ASW or ASWF failures, or parity errors are resolvable on one interrupt, unless the fault is transient.

Three basic forms of hardware redundancy are used in the No. 4 ESS architecture to achieve its fault-tolerant objectives: duplication, memory backup, and *n* for *m* sparing. Major common-control elements and critical memory are fully duplicated: the central processor, variable call memory, disk memory, peripheral controllers, and bus access. The switching network is completely duplicated, including both control elements and switching matrix. These duplicated units are operated either in full duplex mode, with parallel operation and possibly matching, or in active/standby mode, with only one unit active and the other ready to assume the active role on demand.

Figure 1 illustrates the redundancy arrangements for the processor subsystem. All three types of redundancy are used: duplication, with and without matching; simplex with backup; and *n* for *m* sparing. Instruction memory, i.e., the program store, and office data memory, are fully backed up on disk, and only one on-line copy of each is provided. In the event of a failure in the on-line copy of office data, one of the duplicate call-memory stores is renamed to replace the failing store, and its contents are pumped from disk. This is a form of *n* for *m* sparing, where *n* is the number of duplicated call-memory units in the system. For program stores, two spare memory units are provided. These, like the office data replacements, can be renamed to replace any program store unit that fails, and the appropriate instruction memory can be pumped from disk.

Figure 2 shows the redundancy arrangements used in the peripheral subsystem. Here again, all types are represented: duplication, with and without matching; simplex with backup; and *n* for *m* sparing. Units that serve a limited number of trunks or special circuits are protected

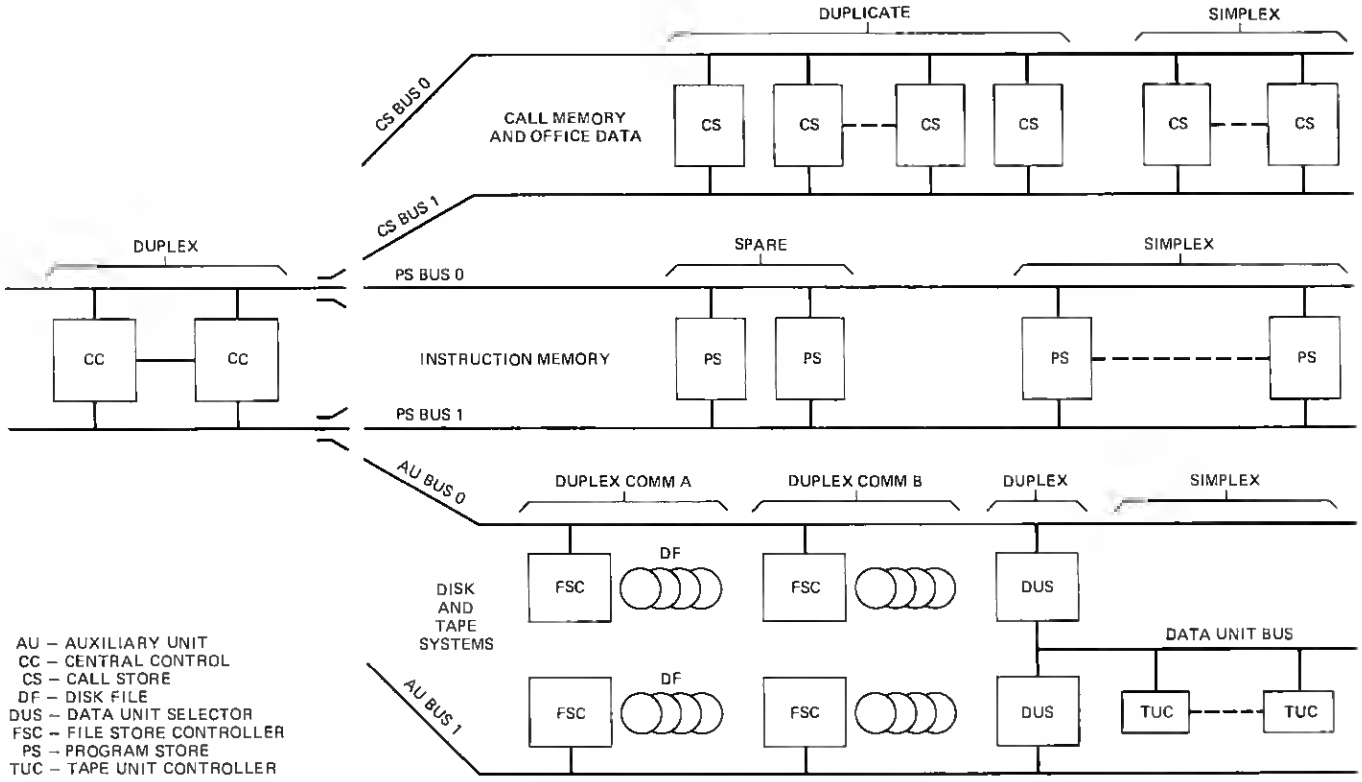

Fig. 1—Processor redundancy arrangements.

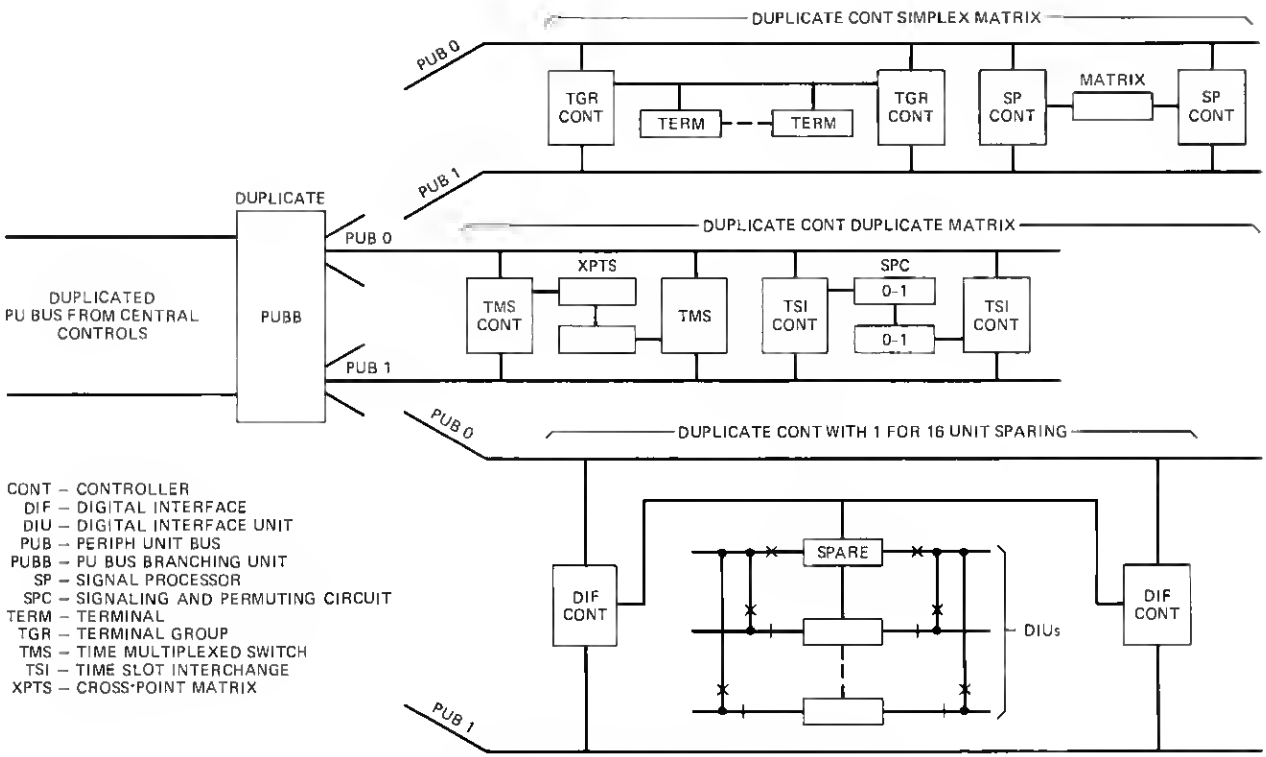

Fig. 2—Peripheral redundancy arrangements.

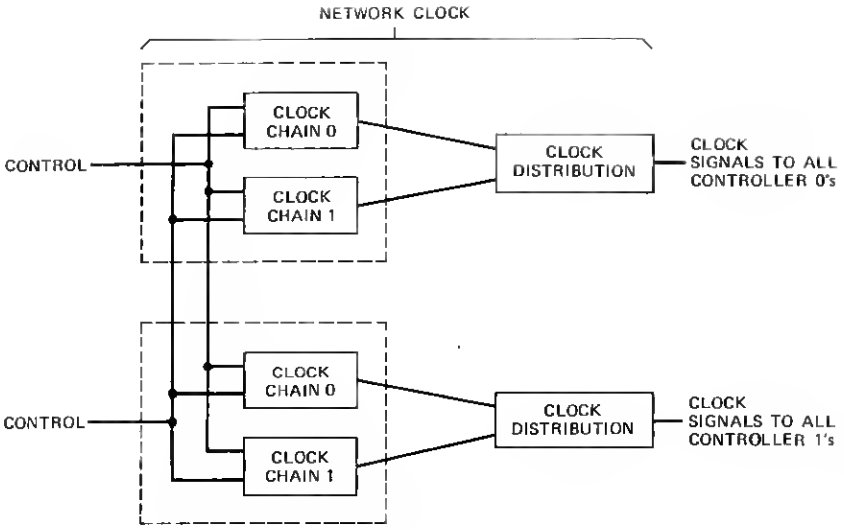

Fig. 3—Network clock (NCLK) redundancy arrangement.

on a limited sparing basis. These generally have one or two spare units that serve as backup for several other units, e.g., one spare digital interface unit (DIU) backs up 16 active units. A spare unit can replace any of the active units by operation of a protection switching mechanism, under control of the fault recovery program system.

The synchronization clock, which is especially crucial to network operation, is quadruplicated. Figure 3 illustrates the special redundancy arrangement used in the network synchronization unit to achieve the high level of fault tolerance required of that unit.

3.2 Software structure

The basic integrity of system memory is protected by audit programs that are structured into mutilation detection and correction modules. The detection modules are constantly run in the background by an audit control program. Additional protection from mutilation is provided by widespread use of defensive checks, e.g., range checks and/or consistency checks in the operational programs. Defensive check failures invoke isolation and corrective responses.

Program integrity assurance is the responsibility of an overseer called the system integrity program. Job scheduling and sequencing are monitored for both frequency and execution times. Program sanity timers are administered. Error conditions are corrected by calling appropriate audits, or by involving various levels of system reinitialization.

IV. ARCHITECTURAL INFLUENCE ON FAULT-TOLERANT DESIGN

The distributed architecture in the peripheral system in the No. 4 ESS poses a unique and often perplexing set of recovery problems. Many peripheral operations are performed autonomously within units that have their own error detection logic. When a fault is autonomously detected, its occurrence is reported to the central processor via a clocked interrogation pulse that triggers a peripheral interrupt. The processor-based recovery control program determines which unit experienced the fault by polling all units using broadcasted interrogation pulses that elicit unique identifying responses from the faulted unit. Once this unit is identified, the appropriate fault recovery actions can be taken. Isolation of these autonomously generated interrupts can be particularly difficult because high levels of subsystem interconnections can cause faults or errors to propagate, and to be reported by other than the unit in trouble. This is especially true in the switching network, where the synchronization clock and switching elements are highly interactive. The network architecture and the associated fault recovery strategies designed to achieve fault tolerance are described in Section V.

Recovery from a fault in the system hardware or from a software error can be viewed conceptually as a three-step operation: detection, isolation, and recovery. Detection is the recognition of the problem. Isolation, in the context used in this paper, is the identification of the failing unit or subsystem. Recovery is the elimination of the fault from the operational system, for example, by reconfiguration or reinitialization. Detection is usually straightforward. Parity errors, ASW-type failures, mismatches, time-outs, and similar triggers stimulate some level of system response such as an immediate interrupt for critical problems, or an interject for less urgent cases. Step two, isolation, is the process of determining the location of the problem. This step is highly dependent on the existence of sufficient indicators to point to the problem area. The particular architecture has a major influence on this isolation process. The third step, recovery, is the heart of the survival process. Here again, architecture is an important consideration in the development of fault-tolerant strategies because it impacts the redundancy plan and the recovery options. Architecture is not the sole consideration, however, because the redundancy plan afforded by the architecture only defines the starting point of the system configuration. The state of the system at the time of an interrupt is equally important, because it tells something of the prior events, and determines what remaining options are available to the recovery strategies. It is useful to look more closely at the architectural influence on fault-tolerant design, particularly as it affects isolation and recovery.

4.1 Effects of architecture on isolation

Isolation of a system problem requires an analysis of the fault indicators used in the detection process, considered in the context of the existing system environment. For a failure in a duplex peripheral unit, the fault isolation program must determine whether one or both peripheral controllers detected the fault, whether both central processors were involved, and whether both access buses were in use. The degree of resolution depends on how much of the available redundancy was on-line at the time of the failure, and, indeed, on the uniqueness of the fault indication. Frequently, an architecture employing real-time matching between duplicate control units will detect a fault without giving a unique indication of which unit half has the fault.

Beyond the question of redundancy, a high degree of interactive coupling exists in the No. 4 ESS, particularly in the switching network. Such tightly coupled architecture poses a real challenge to rapid and effective isolation, particularly if the detection logic does not provide a unique error indication.

Failure modes can affect isolation. This is not in reference to classical gate failures, stuck-at-faults, etc., but failure modes as they may affect accuracy of isolation. For example, a power supply failure, wherein there is high capacitive filtering, can give false indications to the isolation program because of the time constant in the power supply filter. This can occur because some logic gates fail before others as power drops from its proper level towards zero. As another example, an autonomous peripheral unit may trigger an interrupt in the central processor, but then, by the nature of the fault, be unable to respond to the interrogation pulse sent to identify the failing unit. As a further example, there have been experiences where a control unit, whether by timing idiosyncrasies or by the nature of the fault, entered a state where it would not respond to any test commands until the unit was forced into an initial state by physically cycling power on it. These are but a few examples of how failure modes can affect isolation.

Isolation of a software problem is also affected by architecture. Autonomous peripheral units control many operational and maintenance processes, and pass data to the central processor via report buffers. This architecture, using buffers, requires the program monitor system to screen out invalid reports, out of sequence reports, and to correlate the data to the structures that could be implicated. The importance of this issue has become intensified with the addition of peripheral microprocessors and adjunct processors, with their loosely coupled architecture and further delayed buffering of information.

4.2 Effects of architecture on recovery

The issue of recovery, interestingly, is more difficult to resolve than

that of isolation because service objectives and architecture play a strong role in determining whether the resolution can be carried out. Isolation is the process of determining where the fault lies, whereas recovery is the process of removing the faulted unit, if possible. The *if possible* is the key issue in recovery. If the active half of a duplicated unit is the only available half, then removing that half would be tantamount to degrading service. Such action may have to be taken eventually, but service impact, accuracy of isolation, etc., must be considered first.

V. DESIGN OF FAULT-TOLERANT STRATEGIES

The foregoing discussion of the influence of architecture on the isolation and recovery process has only touched upon the real question of how to design fault-tolerant strategies for the No. 4 ESS, or, for that matter, any large real-time, high-availability system.

A basic premise of ESS maintenance is the single-fault assumption. Under normal circumstances the mean time to failure of any part of the hardware is much greater than the time to repair that failure; therefore, no part of the system should experience simultaneous multiple faults that would impair service. From this it follows that fault recovery should be a simple one-dimensional process.

Then why is it not always so? The architectural redundancy plan came from the basic premise. Critical units, including most common control elements, are fully duplicated. Memory is either duplicated directly or backed up in some other medium, e.g., disk and random access stores. Other units, particularly units that interface with trunk circuits, are engineered with one or more switchable spares, determined by reliability considerations and service objectives. Normally, all redundant elements are configured for maximum availability. A classical stuck-at fault occurring within such an environment will normally be detected and uniquely isolated from a single interrupt stimulus.

5.1 Basic duplex recovery strategies

When a duplex unit causes a system interrupt, recovery is effected by isolating and removing the faulted unit from service, and activating the redundant half, consistent with the remaining system environment. This same approach applies to units with backup or spare redundancies. If the redundant unit is available, it will be pressed into service when the active unit fails. The recovery action is concluded by a request for a full diagnostic test of the faulted unit to determine the location of a replaceable module. Once repaired, the unit will again be diagnosed, and returned to duplexed operation, if it passes the diagnostic tests.

The preceding scenario describes the simple process of fault recovery

as it occurs most of the time. Why does it not always happen this way? It is simply because of the vagaries of the system. The single-fault assumption is true most of the time. But, it does not always take a fault to put the system into a simplex mode of operation. Much of the hardware contains maintenance logic used to detect faults. The operational logic is exercised by executing operational commands, but what exercises the maintenance logic? Generally, much of the fault detection logic is unchecked during execution of normal commands. To ensure the integrity of the fault detection capabilities, most of the logic is periodically tested during low traffic periods. During such tests, the unit being diagnosed is usually removed from service, thus foregoing full-duplex operation for the duration of the routine tests.

The frequency of the routine testing takes into account the length of the test, and the residual reliability of simplex operation to the system. Many of the units may be routinely tested on a daily basis, some every few days to a week. While a unit is in simplex operation, recovery from a fault in the active control half requires a different, more subtle, strategy than the simple duplex strategy described earlier.

5.2 Basic simplex recovery strategies

When a unit half is removed from service and a diagnostic test of that unit scheduled, the recovery program will, where possible, try to put the suspect unit into a special out of service mode. In this mode the recovery program's internal memory and sequencers will track the active unit in case the recovery operation had implicated the wrong control unit. Such a mode is referred to as a listen-only mode (LOM). If, before the start of the diagnostic tests, the active unit should interrupt, the LOM control unit can be put back into service immediately, without the need for updating it, and the mate control unit can be removed. Once the diagnostic tests have started, of course, the out of service control unit becomes out of date and cannot be put back into service without updating or reinitializing it.

Consider the first of these two cases just cited. If the fault recovery program removes the wrong control unit on the first stimulus from a mismatch error, the fault in the active unit would likely cause an immediate second interrupt, while the out of service unit was still LOM. The second recovery action would restore the previously removed control unit, remove the active unit, and request diagnostic tests. In the second case cited, with the out of service control unit out of date, a fault in the active unit can have serious service repercussions. Reinitializing the out of service control unit, particularly its memory, will cause the loss of all calls that may be in progress within that unit.

Considering that a unit may be serving as many as 4,000 trunks, or possibly even a quarter of the calls active in the switching network,

reinitialization of a control unit demands serious deliberation. To protect against undue service impact, the basic simplex strategy turns to considering the impact of the present interrupt. Could this be due to a transient fault? How long has it been since the last interrupt? Presumably, if the rate of interrupts is sufficiently low, the system could tolerate an occasional interrupt, and a possible mishandled call, better than it could a major service disruption. To estimate the service impact of the present interrupt, the fault recovery program uses a Leaky Bucket Counter (LBC) to judge the rate at which the interrupts are occurring. For each interrupt that is experienced, the LBC is incremented or decremented by an amount that is determined by the elapsed time since the last interrupt. The higher the interrupt rate, the faster the LBC will reach a predetermined threshold, and reinitialization of the out of service control unit will take place. If the interrupt rate is low, or of short bursts, the LBC may never reach threshold, and the system will tolerate the perturbations.

When a hard or persistent fault causes the LBC to reach its threshold, a reinitialization, or zero-start as it is called, will be requested on the out of service control unit. This action will cause all calls in the unit to be lost. Additional memory reinitialization outside the unit may also be required, as in the case of the switching network with its network maps, and with trunk interface units with the trunk state memory.

The basic simplex strategies provide a high measure of fault tolerance for the No. 4 ESS for hard faults, and use the redundancy in the system for maximum service availability. For software errors, transients, and several other situations that may or may not be unique to the No. 4 ESS, these strategies are, however, found wanting or inappropriate. Further, there are other major considerations that affect survivability: changing fault data, shifts in strategy, false conclusions, etc. These, and other important dimensions, will be considered later in this paper. First, however, the software error and transient strategies will be discussed.

5.3 Recovery from software errors

System problems can be caused by at least two types of software errors: program anomalies (bugs), and bad data. There is a better chance of coping with bad data than with program bugs, but there are recovery strategies in the No. 4 ESS to deal with both.

When a call-handling program tries to address a non-existent unit within the No. 4 ESS, an ASW interrupt will occur because there is no unit to return the expected affirmative response. Presumably, the program that issued the command is working with bad data. The ASW failure triggers the fault recovery process, but in this case, the trouble

is not a fault but a program error. No hardware recovery action is therefore required. The appropriate reaction is to correct the data being used by the call-handling program, or to kill the call, or both if necessary. The recovery strategy will call on the memory audit system to run one or more relevant audit programs to detect and correct memory inconsistencies that may exist. The recovery program will avoid returning control to the program that issued the order, but rather will return to a safer control point. Despite that precaution, it is possible that the same program could regain control at a later time and cause the system to take additional interrupts before the audits can completely correct the problem. To allow the audits a chance to correct the problem, the recovery strategy may decide to inhibit, or prevent, further interrupts of that type. This process is called pesting interrupts.

Not all interrupts caused by bad data occur as a result of invalid commands to non-existent units, causing what is commonly called an out-of-range (OOR) failure. Many interrupts that are in range, i.e., from valid units, can be isolated to software causes. Again, appropriate audits are called by the recovery strategy. One difference, however, is that local- (frame) level pesting of interrupts can be used, rather than the system-level pesting used in the OOR case.

Running pested, or with interrupts inhibited, is risky, at best, because some fault detection capability is disabled. Such risks are justified, however, under the single fault (or error, in this case) assumption.

One can carry the software issue a step further in developing a recovery strategy to meet all cases. There often exists a possibility that the resolution of a software error is, in reality, not true, and that the real problem is indeed a hardware fault. The accuracy of the resolution is not infallible. Using the software recovery strategy described does not take into account the possibility of a hardware fault, and therefore would not resolve the issue if that were true. In this case, at some point in the recovery sequence, one of the involved control units would be removed from service and diagnosed. If neither control unit is found faulty, the final action of pesting the frame would be invoked.

When an interrupt is caused by a program bug and not by bad data, the previous strategies of calling audits and/or removing hardware may not resolve the problem. The final action of pesting the system will, at least, allow continued system operation, albeit in a possibly degraded or more vulnerable environment. In many instances where a program bug has caused problems, running pested has given maintenance personnel enough time to analyze the fault recovery data and make operational adjustments. These adjustments could be in the form of immediate program corrections, procedural changes, or temporary program changes to pin-out or bypass the offending program.

5.4 Basic transient strategies

It is generally easier to deduce that a fault is transient than it is to isolate it. The results of retrying a failing command can usually establish whether a given fault condition is hard or transient. The problem is that when the fault appears to be transient, there is some doubt that the proper suspect has been determined, or if it is the right suspect, the diagnostic tests will find nothing wrong. This leaves the strong possibility that the fault can plague the system for some period of time.

There are two primary strategies for coping with transient faults in the No. 4 ESS, one to cover the case where the isolation program is reasonably certain of the suspect unit, and another to qualify the decision with a measure of uncertainty. In both cases, on the first interrupt, the primary action is to record the error and clear the error conditions in the suspect unit. If a second interrupt occurs, the suspect will be removed from service, provided the duplicate mate, or backup unit, is available. The suspect will be put into the LOM state to ensure that it will remain up to date in the event of a wrong decision. Cyclic diagnostic tests will be scheduled on the suspect unit, expecting that a transient fault might not be caught by a single set of tests. For example, diagnostic tests might be repeated three times, to improve the probability of isolating the fault.

The two transient recovery strategies diverge at the point of the third interrupt. If subsequent interrupts still occur beyond the second interrupt, and the isolation program indicates uncertainty, the recovery strategy will change the suspect, and remove and call for a cyclic diagnosis of the mate control unit.

Note that it is the diagnostic test results that drive the escalation of the recovery strategy. Each time a transient fault escapes detection by the diagnostic tests, the unit is restored to service. The strategy has a built-in control that will limit the number of times the cyclic diagnostics can be scheduled. Beyond that point a suspect unit will be removed from service without further testing. It is left to the skill of the maintenance personnel, using special test facilities, to trace the problem further.

The transient recovery strategies covered above are used when the suspect and its mate are both on-line and available at the time the interrupt occurs. If a transient fault causes an interrupt while a unit is simplex, the basic simplex recovery strategies described earlier apply. When redundancy is not available, the recovery strategy is insensitive to whether the fault is hard or transient.

5.5 Special problems

Several other fault-tolerant strategies have been developed for the

No. 4 ESS to cover situations that are beyond the scope of the more general strategies described above. Buffer overflow, network interface problems, internal network failures, and many other cases require special recovery strategies.

5.5.1 Buffer overflow

Autonomous processors operating asynchronously communicate with the central processor via report buffers. Buffer sizes are engineered to provide enough capacity to absorb spurts in activities and occasional delays in retrieving reports from the buffers. Critical buffers are protected by overflow detection logic that triggers an interrupt, or more likely, a lower priority interject, and calls a fault recovery sequence into action. Possible causes of buffer overflow could be related to program integrity, i.e., the report handler not being scheduled, unusual traffic overload or congestion, or even an undetected hardware problem.

The primary recovery strategy covers two probable causes. It schedules the report dispenser program to unload the hyperactive buffer, and also calls for an audit of the task sequencer control tables. To prevent any further overflow triggers from firing before the recovery action is completed, the buffer overflow detection logic in that unit is pested, i.e., inhibited, for some short period of time. If overflow conditions continue despite the recovery attempts cited, the hardware becomes suspect, and appropriate actions to remove a control unit are attempted, consistent with the strategies described earlier.

5.5.2 Network interface problems

There exists a number of units in the No. 4 ESS that are situated between the switching network and the trunk circuits and that can be classified generically as the network interface units (NIUs). Functionally, these units are responsible for separating the signaling and voice or data information, and converting the voice/data from the incoming or outgoing transmission format (analog or digital) to the pulse code modulation (PCM) format switched by the No. 4 ESS network. The NIUs interface with the switching network via coax cables, each carrying 120 trunks time-multiplexed onto 128 time slots. Several NIUs are protected by one spare unit that can be protection-switched into service by the recovery program. The number of units served by each spare varies with the type of unit in question, but it ranges from one spare for six units to one for 16.

While the NIU recovery problems may be unique to the No. 4 ESS, or possibly other ESS machines, the fault-tolerant strategies developed for this purpose may have relevance for other $1/m$ or n/m redundancy arrangements.

A fault occurring in a NIU can be detected either by the NIU control unit or by the switching network, depending on the location of the fault. Regardless of how the fault is detected, the basic recovery concepts for hard or transient faults apply. The pointed difference in strategy comes about when the recovery decision is to remove the faulted unit. If the spare NIU is available, it will be switched into service, much as is the mate of a duplicated control unit. If the spare NIU is presently serving in place of another NIU, or if the spare is out of service, then the active NIU cannot be removed without degrading service for the 120 trunks served by that NIU. In that event, on the first interrupt, only clean-up action is taken. On later failures, the fault recovery strategy tries to optimize the use of the spare NIU, and if possible move the spare to replace the present suspect NIU. If that cannot be done, then the suspect NIU is removed from service and the 120 trunks are correspondingly taken out of service and marked "maintenance busy."

An interesting extension of the recovery strategies for NIU failures is invoked when multiple NIU errors are detected simultaneously. On multiple errors, the recovery strategy escalates to implicate one or even both of the NIU common control units, in place of, or in addition to, the NIUs themselves.

5.5.3 *Internal network failures*

Perhaps the most sophisticated fault-tolerant strategy developed for the No. 4 ESS involves the isolation and recovery from internal errors in the Time Division Network (TDN). This switching network includes a four-stage space switch with time-switch matrices as initial and final switching stages. The PCM voice/data bit streams are time-multiplexed onto 128 time slots, and are protected by simple parity and a leading-one protocol.

The components of the TDN are two units that provide time-space and space-time switching stages, one for transmitting and one for receiving, located on either side of a unit that provides a two-stage space-switched matrix. The network topology is shown in Fig. 4. The TDN architecture is a multi-dimensional, tigbtly coupled complex. Both the time and space switching units are fully duplicated, including the switching matrix, and are normally operated in a full-duplex, matching mode. Either half of the transmitting or receiving switch units can communicate with either half of the space-switched unit.

A special problem posed to the No. 4 ESS fault recovery system is the problem of resolving the class of internal TDN errors called transmit/receive parity failures (TRPFs). A TRPF is a failure in one of the talking paths, i.e., time slots, caused by a fault somewhere in the TDN matrix elements. On a given TRPF interrupt, six possible sus-

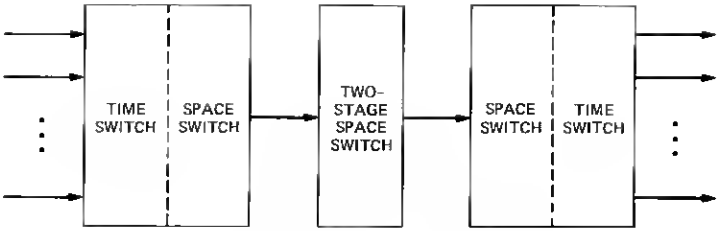

Fig. 4—Time division network topology.

pects exist (one space-switched and two time-space-switched units, duplicated).

In a well-behaved, single-fault situation, i.e., a readily isolated hard fault, the isolation program will determine which elements of the TDN were involved in the failed talking path and set up all possible combinations of links between the various duplicate matrix elements. By a process of elimination, the isolation program can determine which combinations experience an error and which do not. If all combinations are available, and a single fault exists, a unique isolation can be accomplished on one interrupt, and recovery effected by removing the faulty control and matrix half. Again, as with previous strategies, diagnostic tests are scheduled to facilitate repair.

Three conditions can short-circuit this network recovery strategy: the fault is transient (more accurately, marginal), the fault is multiple, or the TDN was not fully duplex at the time of the interrupt. Any one of these conditions prevents unique isolation and accurate resolution on a given interrupt. A special recovery strategy has been developed to cope with this problem.

On each unresolved TRPF interrupt, the recovery program updates special leaky bucket counters for the three (and possibly six) units/matrices involved in the failed talking path. Such records are maintained on each unit in the TDN. The first unit to exceed a predetermined leaky bucket threshold is deemed suspect, and its control unit is removed and diagnosed. The assumption is that calls will be evenly distributed throughout the network, and if any one unit is involved in TRPF failures above all others, then the weight of suspicion falls on that unit. Diagnostic tests will confirm or deny that suspicion.

5.6 Practical considerations on total fault-tolerant design

All the fault-tolerant strategies previously considered addressed specific problems of isolation and recovery. There is a whole area of concern in the design of a total fault-tolerant system beyond that already discussed.

5.6.1 Inconsistencies in fault resolution

Questions arise about the continuing strategy to use if the fault signature or resolution changes as various recovery attempts are made. How long should the failure history actively be maintained on a unit, particularly after a fault apparently has been repaired?

In the deductive process of isolating a fault in the No. 4 ESS peripheral system a tree-branching structure of decisions is used based on the initial error source data and the results of tests, with the resolution being declared a point of maximum definition (PMD). Theoretically, the PMD data developed from these tests yields a unique identification of the faulted unit, the class of fault (e.g., hard-unique/non-unique, transient, software), and whether the fault was resolved or not resolved. It then gives a recommendation of the strategy to be followed for recovery. The recommendation is based on the current interrupt and current circumstances, with no prior events taken into consideration. The isolation (PMD) data and recommendation are passed to an error analysis program that has access to history records of previous interrupts and recovery actions. If the current interrupt is the first such occurrence, the error analysis program will allow the recommended recovery strategy to be carried out, insofar as it is consistent with service criteria. If, however, a previous history of interrupts exists for the suspect unit, it is important to determine if the current recommended strategy is the same as that followed on previous interrupts. If it is not, which strategy is it appropriate to follow?

There are many reasons why the resolution, and consequently the recommended recovery strategy, can change from one interrupt to the next. The fault could be data sensitive, the error indications could be sensitive to the state of the unit, or the problem could even be an actual or apparent multiple fault situation. In any event, there are cases where the previous recovery strategy should be continued, and some where the current recommendation is more appropriate. Also, sometimes the current PMD data could preclude a continuation of the previous strategy because of data and/or environment incompatibilities.

A common situation would be for a transient fault to later develop into a hard fault. Different strategies for recovery are followed for each case, as described earlier. In this instance, it is reasonable to change from the transient strategy to a hard-fault recovery sequence because the latter addresses a specific rather than a nonspecific causal relationship. If the reverse were true, that is, if the resolution of a particular interrupt indicated that the fault was transient when all previous experiences were seen to be hard, then it would probably be inappropriate to abandon the previous strategy. Many conditions could alter

this position, however. Is it the same fault in each case? Had repair been attempted prior to the latest interrupt? How much time had elapsed since the previous interrupt?

The point of these questions is to determine if two successive interrupts are caused by the same fault, when the resolution in each case might suggest otherwise. Because of these natural state transitions (transient/hard), and for countless other causes, it is necessary to recognize when different recovery strategies are indicated on successive interrupts and to determine which of the two strategies to follow.

Before the process used in the No. 4 ESS for escaping between recovery strategies is described, an explanation of the process for recognizing the existence of prior associated events and for comparing the past and current recovery strategies will be given.

5.6.2 Error Analysis

The isolation of a fault or other cause of an interrupt is under the control of the Fault Recovery (FR) programs. As explained earlier, the FR programs isolate the fault or error using a deductive process consisting of retries or specific tests under various configurations. The results of these tests are passed to the failure error analysis (FERA) programs as PMD (point of maximum definition) data, detailing the identity of the suspect unit or cause, the class of fault, and indeed, whether the problem was resolved. The data further includes a recommendation of a recovery strategy to follow. Keep in mind this PMD data is based strictly on the current problem. The FERA programs represent an extension of the recovery process. As each interrupt is processed, a record is retained of the PMD data and the final action taken. When an FR program isolates the cause of a current interrupt, FERA searches all active history records for data implicating the same unit. Once found, the past and present recovery strategies are compared with entries in a strategy escape table. If the strategies are the same, or if the previous strategy is still more appropriate, escape will be denied, and FERA will continue to follow the existing recovery strategy. If the new strategy recommended by the FR isolation program is appropriate to the fault class and status of the suspect, the escape data will allow FERA to abandon the previous strategy and start the new recovery sequence.

The question of escaping from one strategy to another is not solely an issue of changing resolutions, but also can be an issue of changes in the unit state. For example, an FR resolution might call for the removal of a suspect unit, and under a duplex unit availability, FERA could concur. Under a simplex unit configuration, however, the same recommendation, if followed, would cause service degradation. The escape table data will cause FERA to deny the initiation, or the continuance,

of a duplex strategy, when a unit is no longer configured in duplex. In such a case, an alternative strategy is not necessarily provided in the PMD data. FERA must make that determination from strategy selection tables provided for that purpose.

The issue of whether successive interrupts from the same unit might be caused by the same fault is covered grossly by timing out an active history file if the elapsed time since the last interrupt exceeds some threshold, e.g., fifteen minutes. The premise is that most faults in the No. 4 ESS will stimulate an interrupt within that time period, until recovery is effected. Interrupts occurring at greater intervals have a high probability of independence. After repair has been made, and the unit appears fault free, the time-out interval should be sufficiently short to permit a subsequent fault in that unit to be treated with a new recovery sequence. When an active history file has been timed out, it no longer influences subsequent interrupts from a given unit. A record of the events up to that point are, however, kept in long-term storage for manual analysis, if needed.

The mobility provided by the escape mechanism, and to a lesser degree, by the time-out of active history files, permits the error analysis process to cope with possible inaccuracies in resolution, changing fault signatures, and the occurrences of apparently disassociated events.

VI. EXPERIENCE AND EVALUATION

To date, a total of 38 different recovery strategies have been developed for the No. 4 ESS peripheral maintenance system to handle a wide range of fault conditions and software problems involving more than a dozen types of units. Fifteen of these strategies serve the common needs of several units, and twenty three are specialized recovery sequences for particular units or for unique fault situations.

New recovery sequences are developed for each new unit or important new feature added to an existing unit. Recovery sequences might require changes as units are redesigned for cost reduction. System exposure and field support activities provide important feedback to the ongoing process of evaluating and improving the performance of the recovery system.

6.1 Adaptability

A very important attribute to build into the design of a fault-tolerant system is adaptability, i.e., the ability to change as experience and requirements dictate. The initial design of the recovery strategies and control programs in the No. 4 ESS did not adequately take into account the need to switch strategies in midstream of a recovery sequence. Alternate strategies frequently were not available or at least

not specified by the isolation programs; and even when available, the escape control mechanism was distributed throughout the many recovery programs, and was difficult to change. Program changes were cumbersome to make, if not with high risk. Through experience and evolution, the No. 4 ESS peripheral fault recovery system has become highly structured and has matured into a system that has a high degree of adaptability. The escape data allows the error analysis process to either escalate a previous strategy, or to discontinue the previous and invoke a new strategy, as successive attempts are made at recovery. This feature has given the No. 4 ESS the flexibility to take full advantage of experience.

Strategies are developed based on a knowledge of system requirements and predicated on an expected system behavior under a given fault condition. Frequently, however, one learns that when the hardware and software are brought together for testing, the system behavior is not exactly as expected. Whether the cause is the accuracy of detection or accuracy of resolution, or a combination of both, adaptability becomes essential. This is particularly important when recovery problems are exposed in a software/hardware package deployed in switching systems carrying live traffic. It must be possible to develop and test program changes and deploy them in the field as quickly as possible, with minimum risk to the quality of service and to the integrity of the overall system. This has been a major goal in the development of the fault-tolerant No. 4 ESS switching system.

6.2 Experience

The performance of a fault tolerant system can be measured by its track record in encounters with hardware failures and other system troubles.

Software deficiencies and coding errors will always exist in a large system. It is not cost-effective and probably not possible to test a system to the point where all problems have been found and removed. What is important is that service-affecting incidents are kept to a minimum, and that any such occurrences are analyzed for cause. Program corrections or enhancements are developed thereupon, when appropriate.

The level of fault tolerance demanded for the No. 4 ESS is to detect a failure and isolate and recover from that failure with minimum disturbance to call processing. A well-behaved hard fault, i.e., one that reacts consistently under test, can generally be isolated on a single stimulus, and recovery can be effected immediately if redundancy is available. Transients, or other faults that cannot be uniquely resolved, can reasonably require two or more interrupts before they are successfully isolated. These are reasonable levels of performance.

If a fault occurs at a time when the redundant unit is not available, i.e., when the mate unit is out of service either for routine testing or an actual fault, the current fault represents a duplex failure situation. The resolution may be clear, but the recovery cannot be executed without disrupting service. Under these circumstances, an attempt is made to tolerate the problem as long as possible. That is the purpose of the leaky bucket strategy, which estimates how well the system can tolerate the fault by measuring the frequency of the interrupts. A low rate of interrupt can be tolerated longer than a high rate. The actual attempt at recovery will occur at the point that the leaky bucket threshold is exceeded. This action will generally have a disruptive effect on service. The number of interrupts required to reach that point is, of course, a function of the rate of the interrupts.

Since the introduction of the No. 4 ESS in January 1976, seven program generics or versions have been developed and put into general service in more than 75 offices. While the primary impetus for a generic development has been to add new features or major cost reductions, many improvements have been introduced in the recovery system in each generic, in the form of corrections or design enhancements. Most of these were stimulated through laboratory testing, and through feedback from the field. Threshold parameters have been fine tuned in response to field experiences. Several strategies were made more tolerant of faults, and others were rendered more sensitive.

The principal goal of the No. 4 ESS fault-tolerant system is to recover from any fault with minimum impact on service and with a minimum number of interrupts. When recovery is possible, no strategy should inadvertently cause service degradation, e.g., any unnecessary zero-start or duplex failure actions. All service-affecting incidents are analyzed for cause by a field support team. Of the total number of incidents causing service outages for the period January through June 1981, less than ten percent were judged to be caused by deficiencies in the peripheral maintenance software system being discussed herein, whereas hardware problems and procedural errors accounted for almost 50 percent of the incidents.

VII. SUMMARY

The fault-tolerant design of the No. 4 ESS has been evolving since its introduction. Sophisticated recovery strategies have been developed to give the system the necessary high level of fault tolerance. The incident and recovery data printouts have been designed to assist the support team in making analysis of a problem fast and accurate. Fault isolation can implicate multiple units in some situations where a unique resolution is not possible. Multiple recovery actions can be carried out on a single stimulus.

At the current state of the art, the No. 4 ESS is a high-performance switching system, with a successful record for reliability and availability. Its fault-tolerant design has contributed significantly to that success.

REFERENCES

1. J. J. Kulzer, "Systems Reliability—A Case Study for No. 4 ESS," INFOTECH State of the Art Conference on Computer System Reliability, London, England, June 1977.
2. M. N. Meyers, W. A. Routt, and K. W. Yoder, "Maintenance Software," B.S.T.J., *56*, No. 7 (September 1977), pp. 1139–67.
3. P. K. Giloth and H. E. Vaughan, "Early No. 4 ESS Field Experience," Int. Switching Symp., Kyoto, Japan, October 1976, pp. 241-4-1–7.
4. P. K. Giloth, "No. 4 ESS Reliability and Maintainability Experience," 1980 Proc. Ann. Reliability and Maintainability Symp., San Francisco, CA, January 22–24, 1980, pp. 388–92.